A SOA Repository with Advanced Analysis Capabilities
Improving the Maintenance and Flexibility of Service-Oriented Applications

Thomas Bauer[1], Stephan Buchwald[2], Julian Tiedeken[3] and Manfred Reichert[3]
[1]*Neu-Ulm University of Applied Sciences, Dep. Information Mgmt, Neu-Ulm, Germany*
[2]*T-systems International Gmbh, Systems Integration - Car2x Solutions, Ulm, Germany*
[3]*University of Ulm, Dep. Database & Information Systems, Ulm, Germany*
thomas.bauer@hs-neu-ulm.de, stephan.buchwald@t-systems.com, {julian.tiedeken, manfred.reichert}@uni-ulm.de

Keywords: Service-Oriented Architecture (SOA), SOA Repository, as-Is Analyses, What-if Analyses.

Abstract: In a service-oriented architecture (SOA), a change or shutdown of a particular service might have a significant impact on its consumers (e.g., IT systems). To effectively cope with such situations, the IT systems affected by a service change should be identified before actually applying the latter. For this purpose, a SOA repository with advanced analysis capabilities is needed. However, due to the numerous complex inter-dependencies between service providers and consumers, it is a challenging task to figure out which IT systems might be directly or indirectly affected by a service change and for which period of time this applies. The paper tackles this challenge and presents the design of an advanced SOA repository enriched with analysis capabilities. In particular, this repository enables automatic analyses to detect already existing problems (as-is analyses) as well as problems that might occur due to future service changes (what-if analyses). Respective analyses will foster the development of robust service-oriented applications.

1 INTRODUCTION

Major goals of a service-oriented architecture (SOA) are to increase the reuse of services as well as the flexibility to adapt services to changing business needs (Josuttis, 2007). The principle of loosely coupled services (Erl, 2005), for example, enables a rapid migration of a service implementation to a new version or the replacement of a service by another one, without need for any adaptions on the side of the service consumer. Usually, in a SOA a proxy (i.e., Enterprise Service Bus, ESB) is introduced between the application that calls (i.e. consumes) the service and the service itself. Consequently, the service can be simply exchanged by adapting the proxy accordingly, but no additional adaptations of the consuming application become necessary.

In a case study we conducted in the automotive industry ENPROSO[1], we analyzed a multitude of techniques to increase the flexibility of a SOA (Buchwald et al., 2011; Buchwald, 2012). In this context we learned that other kinds of changes of operable services might be more complex than the aforementioned ones. Especially, this applies to the shutdown of a service (i.e., the service shall no longer be operable), which might affect the operability of the service consumers' applications (i.e., IT systems). Note that it is crucial to detect such outdated services in order to avoid malfunctions regarding the IT systems of previous service consumers (i.e., to avoid the invocation of these services). Besides this, managing outdated services over a longer period of time also causes unnecessary costs on the side of the service provider. For these reasons, techniques are required that allow identifying unused services in a SOA environment (e.g., to switch them off). In literature (Erl, 2005; Josuttis, 2007), the method usually applied for this is to utilize information from a SOA repository (i.e., a service registry or directory). In the ENPROSO project, we developed a comprehensive meta-model for such a SOA repository (Buchwald, 2012). It offers, for example, detailed information on service versions, service installations, and service contracts. Based on this information, in turn, it becomes possible to detect the actually used service versions and installations, the period during which they shall

[1] Enhanced Process Management by Service Orientation, performed at and funded by Daimler AG

be used, and the service installations that are expendable or may be switched off in future. In turn, this allows estimating the efforts required to adapt the applications consuming that service.

The SOA repository we developed contains various kinds of object types as well as their relationships. Usually, users might be lost when being confronted with the large data volume of a SOA repository. Hence, advanced data analyses capabilities are needed in order to be able to detect potential problems that might occur in future (e.g., when a service will be finally shut down). Based on the data from a SOA repository, it should be possible to automatically detect problems that will occur in future. Such automatic analysis capabilities have not been sufficiently addressed in current SOA literature so far. Amongst others, this paper presents advanced techniques to identify services whose shutdown date will be earlier than the expiration date indicated in their usage contracts. In particular, such *as-is-analyses* allow identifying problems based on the current repository data.

As an extension, a SOA repository should also enable *what-if-analyses*. A relevant use case for them is the selection of an appropriate date for a future service shutdown. In particular, in order to decide on this date, we need detailed information about the consequences of the service shutdown depending on the point in time it will be performed; i.e., for different points in time, the service consumers affected by this change as well as the related problems need to be automatically determined (*impact-analyses*). This way, possible solutions of future problems (i.e., missing service installations) can be identified as well. Note that without proper tool support, such what-if-analyses would require high manual efforts by users. Hence, as a second contribution, this paper presents techniques for automated what-if-analyses in a SOA.

Section 2 gives insights into our meta-model for SOA repositories, which provides the foundation of our approach. Section 3 then describes examples of analyses that can be based on current repository data (as-is-analyses). What-if-analyses and the related impact-analyses are presented in Section 4. Section 5 discusses related work, whereas Section 6 summarizes the paper and gives an outlook on future work.

2 A META-MODEL FOR SOA REPOSITORIES

To elaborate the demands of a SOA repository in a practical environment, we conducted respective requirement analyses in multiple business units of a large automotive company (Buchwald, 2012). First of all, this revealed the need of the aforementioned repository analysis techniques. In addition, it became evident that a SOA repository needs to be able to manage a multitude of object and relationship types. Fig. 1 depicts the meta-model we developed as foundation for enabling advanced analyses in a SOA repository.

The meta-model comprises object types covering the business level (left part) as well as the technical level (right part). Thereby, a multitude of object types are considered at the two levels[2], e.g., *Business* and *Technical DataObjectVersions*. Respective information may help users in searching for required services and browsing in the repository (cf. Bauer et. al., 2013). Overall, the provided meta-model covers a large part of the information typically required in the context of a SOA. Depending on the respective company and its SOA philosophy, however, additional object and relationship types may be added or non-relevant types be dropped. In the following, we focus on explaining those object and relationship types, which are required to understand the advanced analyses presented in Sections 3 and 4. For a detailed presentation of all meta-model entities we refer to a technical documentation (Buchwald, 2012, (Tiedeken, 2010).

The central object of any SOA is the *Service*. In order to be able to reflect its evolution over time, a service may bundle different service versions. In turn, the latter may have different input and output parameters or comprise different operations. In particular, the different versions of a service might not be compatible regarding their usage by service consumers. In general, one can distinguish business service versions (cf. *BusinessServiceVersion*, together with its relationship type *isBusinessServiceVersion* in our meta-model) and technical service versions (cf. *TechnicalServiceVersion*, together with its relationship *isTechnicalServiceVersion*).

The implementation and rollout (i.e. deployment) of a technical service version results in a service installation (*ServiceInstall*) – this link is established

[2] Note that it is not necessary to acquire all data stored in the SOA repository manually; i.e., numerous objects may be imported automatically from already existing IT systems, e.g., business process modeling tools, software development environments, or tools supporting government processes. Taking this into account, the maintenance of a SOA repository can be accomplished with acceptable efforts from the user side.

based on relationship *hasInstallation*. Since a service installation is offered by a concrete IT system, it will be connected to this *System* using the relationship *providesServiceInstall*. In turn, relationship *hasProvider* informally describes that the IT system offers this service. In general, a service may be offered by different IT systems and may possess multiple installations that belonging to the same or different technical service versions.

Before a service may be actually used, a *Contract* must be made between the provider and the consumer of the service. In turn, this contract is assigned to the consumer using relationship *hasContract*. Furthermore, the information related to this contract is usually created step-by-step. First, it is determined which business service version shall be used (cf. relationship *hasBusinServiceVersConsumer*). At this point, no

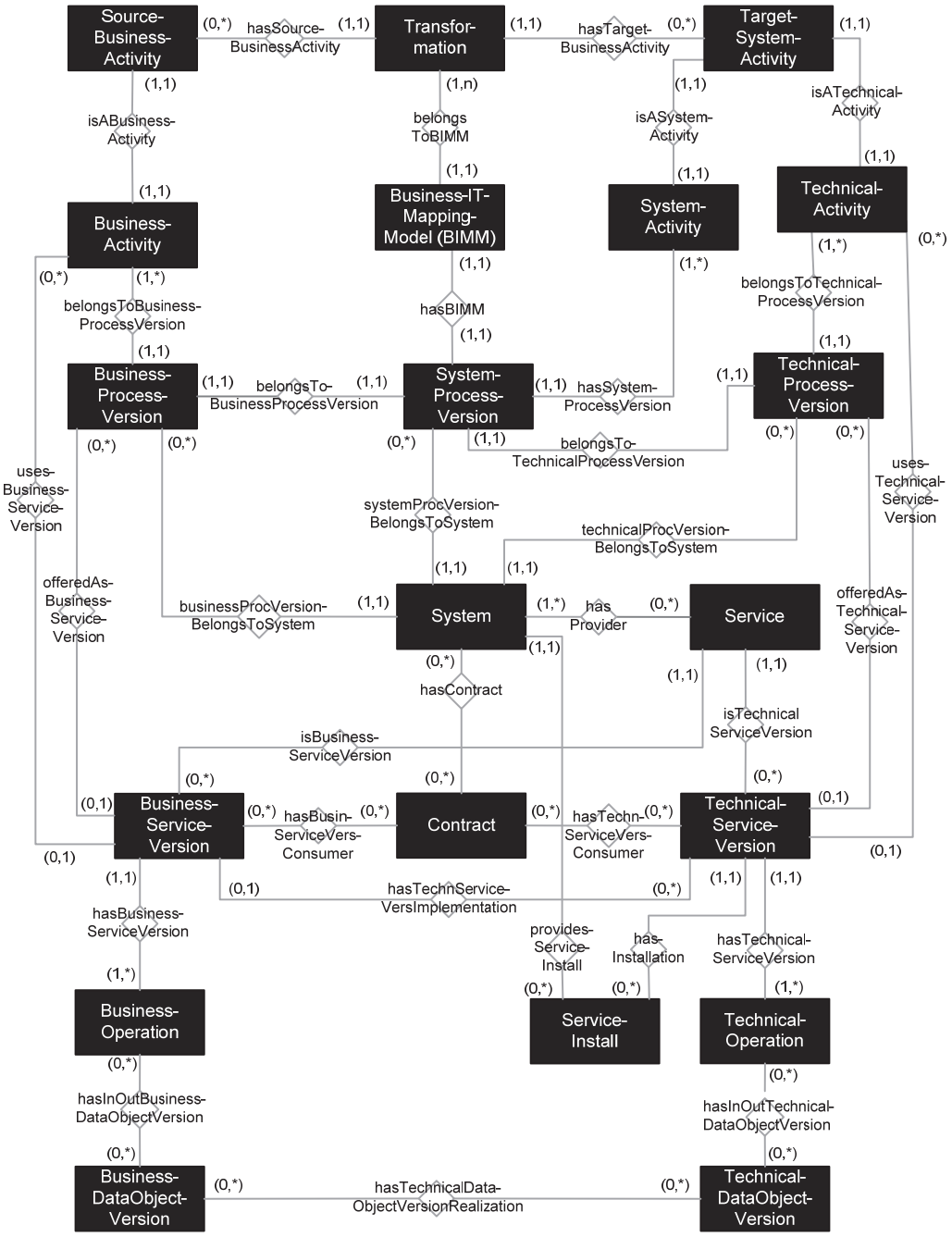

Figure 1: Meta-model of the SOA Repository expressed as Entity-Relationship Diagram (Abrial, 1974).

corresponding technical service version needs to be provided. Later on, a technical service version may be added and assigned to the business service version using relationship *hasTechnServiceVersConsumer*. Additionally, contract data may be enriched with technical service level agreements (Erl, 2005; Josuttis, 2007).

The entity and relationship types used in this paper have the following notation: $e \in E$ holds if entity e is stored in entity set E in the SOA repository. For relationship type b, which links entity types E and E', with $e \in E$ and $e' \in E'$ function $b(e, e')$ returns true if a relationship between e and e' is stored in the SOA repository.

3 AS-IS-ANALYSES

We have already motivated the practical need of as-is-analyses. In particular, the latter can be based on the data captured in the SOA repository. Since the begin and end dates of service usage contracts may be in the future, however, it is not only possible to identify already existing problems, but also to predict future ones. Hence, as-is-analyses allow identifying and solving potential problems in advance. This section presents representative use cases for as-is-analyses based on a SOA repository.

3.1 Incomplete Technical Services

To identify a technical service version for which no installation exists constitutes our first scenario for applying an as-is-analysis. Note that this scenario is practically relevant since the version of the technical service cannot be used yet and hence it has to be handled by the SOA governance board (Erl, 2005). The latter may then decide that the technical service version must be implemented and installed (i.e., rolled out). Alternatively, it may decide that the technical service version is no longer required and hence shall be deleted from the SOA repository. In general, the data required for this as-is-analysis can be retrieved from any SOA repository that is based on the meta-model presented in Sect. 2. More precisely, corresponding technical service versions TSV can be determined as follows:

$TSV = \{tsv \in TechnicalServiceVersion \mid \nexists\, si \in ServiceInstall \text{ with } hasInstallation(tsv, si)\}$

Another undesired scenario occurs if the SOA repository contains technical service versions without associated business service versions. This scenario might occur, for example, if a third-party system contains pre-build services (or service interfaces) or if a migration of legacy systems to a SOA was not performed properly. In such cases, the missing business service versions should be added to the SOA repository afterwards. As a potential benefit of this additional information, the available technical service versions can be easier retrieved by domain experts (cf. Bauer et. al. 2013), which, in turn, increases service reusability. Based on the defined meta-model, the technical service versions without associated business service version can be determined as follows:

$TSV = \{tsv \in TechnicalServiceVersion \mid \nexists\, bsv \in BusinessServiceVersion \text{ with } hasTechnServiceVersImplementation(bsv, tsv)\}$

3.2 Unusable Service Versions

Similar to the scenarios we considered in Section 3.1, the presented meta-model can be used to determine both business and technical service versions in a SOA repository that must currently not be used. Reasons for the latter may be, for example, the absence of a certain service contract or the expiry of a service contract's validity duration.

Based on the described meta-model, for each *Service* we can identify the related *BusinessServiceVersion* by using relationship *isBusinessServiceVersion* (cf. Fig. 1). Regarding the scenario from Fig. 2, for instance, for *ServiceX* the business service versions *BSV1*, *BSV2,* and *BSV3* can be identified. While for *BSV1* and *BSV2* a corresponding technical service version exists, this does not apply to *BSV3* (cf. Scenario 3c described below). More precisely, *BSV1* is related with the technical service versions *TSV1a* and *TSV1b*, and *BSV2* with the technical service versions *TSV2a* and *TSV2b*. Similarly, one can detect that TSV1b has no related service installation. Therefore, this technical service version must not be used at the present moment (cf. Scenarios 3a + 3b).

Which problems might be caused by unusable versions of a business service or technical service? And which actions shall be performed to cope with these problems? Based on the presented meta-model, these and related issues can be resolved through automated repository analyses. However, user interactions will still be required to deal with the problems discovered. For example, selecting a proper action for problem resolution might constitute a task to be accomplished by a SOA governance board.

We consider the following scenarios:

Scenario 1 (No valid contract exists for an

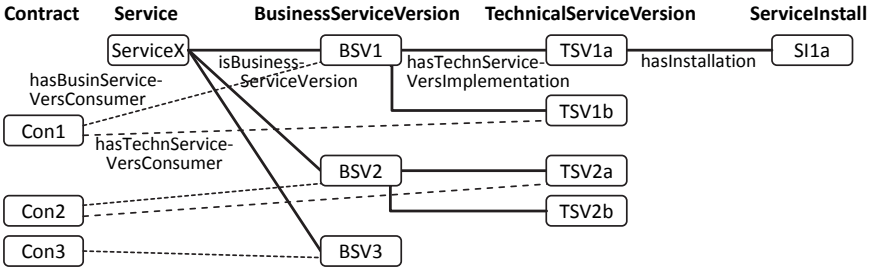

Figure 2: Sample data of the SOA repository belonging to contracts, services, etc.

unusable business service version): Consider Fig. 2: Obviously, business service version *BSV2* must not be used by any service consumer since it has no related service installation (indirectly assigned via *TSV2a* or *TSV2b*) at the moment. If there exists no valid contract for *BSV2* (e.g., in case the expiry date of *Con2* was already passed), this scenario does not result in problems on the side of the service consumers. In any case, if *BSV2* cannot be used, this needs to be properly handled on the side of the service provider. On one hand, it may decide that a service implementation shall be provided for either *TSV2a* or *TSV2b*. On the other, *BSV2* may be declared as unsuitable; i.e., it shall be shut down and deleted from the repository, including the corresponding technical service versions *TSV2a* and *TSV2b*.

Scenario 2 (No valid contract exists for an unusable technical service version): In Fig. 2, the technical service version TSV1b must not be used since it has no related service installation. If there is no (currently) valid contract for *TSV1b* this will not constitute any problem for the service consumer. Regarding the service provider, the governance board may decide whether a service installation shall be created for *TSV1b* or this technical service version shall become obsolete (e.g., if it offers no or only negligible advantages compared to the technical service version *TSV1a* for which a corresponding service installation exists.

Scenario 3 (There is a valid contract for an unusable business / technical service version): The more interesting case occurs if the usage periods of the contracts (cf. Fig. 2) have not expired yet. Then valid contracts exist for service versions that must not be used or whose use might cause severe problems on the side of the service consumer. Regarding this scenario, we distinguish between cases for which an alternative service version exists and cases for which this does not apply:

Scenario 3a (Unusable technical service version with alternative usable technical service version): In Fig. 2, contract *Con1* references *TSV1b*. In turn, this implies that there exists a consumer for *TSV1b* (or will exist in future). Since *TSV1b* does not have a related service installation, the contract cannot be fulfilled. However, there exists another usable technical service version *TSV1a* (with related service installation *SI1a*) assigned to the same business service version *BSV1*. In general, such technical service versions *TSV* may be detected automatically based on repository data with the following criteria:

$$
\begin{aligned}
TSV = \{ & tsv \in TechnicalServiceVersion \mid \\
& \exists\, con \in Contract \text{ with} \\
& \quad hasTechnServiceVersConsumer(con, tsv) \wedge \\
& \nexists\, si \in ServiceInstall \text{ with } hasInstallation(tsv, si) \wedge \\
& \exists\, bsv \in BusinessServiceVersion \text{ with} \\
& \quad hasTechnServiceVersImplementation(bsv, tsv) \wedge \\
& \exists\, tsv' \in TechnicalServiceVersion \text{ with} \\
& \quad hasTechnServiceVersImplementation(bsv, tsv') \wedge \\
& \exists\, si' \in ServiceInstall \text{ with } hasInstallation(tsv', si')
\end{aligned}
$$

To still fulfill the contract, it is possible to switch from the unusable service version *TSV1b* to the usable service version *TSV1a*; e.g., in case *TSV1b* has no real advantages compared to *TSV1a*. In this case, contract *Con1* must be modified as well as the application on the consumer's side (if already existing). Alternatively, it may be decided that a service installation must be provided for *TSV1b*. Furthermore, *TSV1a* and *TSV1b* might only show small differences (e.g., different but similar data structures as input or output parameters). In this case, a service installation for *TSV1b* can be realized by calling *TSB1a* via a proxy (ESB), which eliminates the differences between the consumer and the service with an appropriate transformation of the data structures. Note that such a solution uses the flexibility offered by a SOA (Buchwald et al., 2011).

Scenario 3b (Unusable technical service version without alternative usable technical service version): In Fig. 2, *Con2* references the unusable technical service version *TSV2a*. The sole technical service version TSV2b, which is also

related with BSV2, cannot serve as alternative since it does not have a service installation.

In general, such technical service versions *TSV* can be detected as follows:

$$
\begin{aligned}
TSV = \{ & tsv \in TechnicalServiceVersion \mid \\
& \exists\, con \in Contract \text{ with} \\
& \quad hasTechnServiceVersConsumer(con, tsv) \wedge \\
& \nexists\, si \in ServiceInstall \text{ with } hasInstallation(tsv, si) \wedge \\
& \exists\, bsv \in BusinessServiceVersion \text{ with} \\
& \quad hasTechnServiceVersImplementation(bsv, tsv) \wedge \\
& \nexists\, tsv' \in TechnicalServiceVersion \text{ with } (\\
& \quad hasTechnServiceVersImplementation(bsv, tsv') \\
& \quad \wedge \exists\, si' \in ServiceInstall \text{ with} \\
& \quad hasInstallation(tsv', si') \;)
\end{aligned}
$$

Again, for this scenario a service installation could be added for *TSV2a*. The choices to avoid this, however, are limited compared to Scenario 3a, since there exists no alternatively usable technical service version belonging to the same business service version. Hence, switching to another business service version remains the only solution in this context (i.e., *BSV1* in Fig. 2 since *BSV3* has no related service installation). Switching to *BSV1* could be realized, for example, by modifying *Con2* as well as the application of the service consumer (if required). Typically, for modifying the latter, much more efforts will be required compared to Scenario 3a. In this context, note that business service versions usually show greater differences among each other compared to differences of the versions of a technical service.

Scenario 3c (Business service version without technical service version): Consider Fig. 2. For contract *Con3*, there only exists a business service version, but no corresponding technical service version. Such a scenario may occur, for example, if the realization of the technical service version is not finally decided or its specification is not completed yet. In general, the respective business service version cannot be used before realizing a corresponding technical service version. Alternatively, as discussed in the context of Scenario 3b, one may switch to another already usable business service version (i.e., *BSV1*). Due to lack of space, we omit the formal criterion for detecting Scenario 3c.

4 WHAT-IF-ANALYSES

The analyses described in Sect. 3 have been solely based on data currently stored in the SOA repository. This section complements these *as-is-analyses* with selected examples of *what-if-analyses* (also denoted as *impact-analyses*). In particular, the latter analyze the impact of changes that may happen in future. For example, it can be determined which service consumers will be affected by a service shutdown that takes place at a future point in time. Based on respective information, in turn, the most appropriate time for a service shutdown can be determined. In general, *what-if-analyses* allow simulating and evaluating a variety of scenarios. The resulting information then serves as basis for decision making (e.g., by a governance board).

4.1 Indirect Dependencies

The importance of *impact-analyses* results from the fact that, for example, the shutdown of a service installation not only affects its direct consumers. In fact, these consumers may be IT systems (cf. *System* in Fig. 1) that themselves offer services to other consumers. Consequently, if such an IT system needs to be adapted (e.g., due to a migration to an alternative usable service), the services it offers may have to be changed as well. Even worse, in case there is no alternatively usable service, the concerned services must no longer be provided. In both cases, the consumers of the IT system are affected as well; i.e., the change (service shutdown) may have to be propagated over multiple levels to directly or indirectly dependent IT systems (*ripple-effect*).

Based on the presented meta-model (cf. Sect. 2), such indirect dependencies can be detected automatically. Consider the example from Fig. 3 and assume that a shutdown of service installation *SI1* is planned for the near future. Assume further that *SI1* is the only installation of the technical service version *TSV1*. Then, the shutdown will affect contracts *Con1a*, *Con1b* and *Con1c*. As a result, IT systems *Sys1a*, *Sys1b* and *Sys1c* will no longer work properly and must therefore be adapted. In turn, this might lead to changes of services *Serv2* and *Serv3* that are offered by *Sys1b* and the corresponding technical service versions *TSV2* und *TSV3*. In summary, the original change needs to be propagated along multiple artifacts; i.e., the dependent contracts *Con2*, *Con3a* and *Con3b* must be adapted as well as the IT systems filling the role of a consumer with respect to these contracts. In turn, these IT systems themselves offer services and hence must be adapted as well (and so forth).

In general, when turning off or modifying a service installation *si*, the affected IT systems can be

automatically calculated as follows:[3]

AffectedSys(si) = {*sys* ∈ *System* |
∃ *tsv* ∈ *TechnicalServiceVersion* with
hasInstallation (*tsv*, *si*) for which applies:
∃ *con* ∈ *Contract* with
hasTechnServiceVersConsumer(*con*, *tsv*) and
hasContract (*sys*, *con*) }

All other IT systems indirectly (i.e., transitively) affected by the change can then be identified as follows: For each system *sys* ∈ *AffectedSys(si)*, the offered service installations *AffectedServiceInstall* are determined based on relationship *providesServiceInstall*. In turn, for each service installation *si'* ∈ *AffectedServiceInstall*, *AffectedSys(si')* is calculated on the next level based on the above criteria. This procedure is continued until all affected IT systems are determined.

Note that when solely considering the relationships between IT systems (i.e. System) and service installations (i.e. ServiceInstall), we might identify dependencies that actually do not exist; i.e., from the fact that the program code of an IT system calls a particular service, we must not conclude that all offered services are actually available. In the above example (cf. Fig. 3), system *Sys1b* and, therefore, the offered services *Serv2* und *Serv3* are connected with service installation *SI1* that shall be shut down. In this context, we need to distinguish two cases:

- *Case 1:* Assume that the implementation of *TSV2* and *Serv2*, respectively, is based on *TSV1* and hence on *SI1*. If the adaptation of *Sys1b* results in a changed interface service *Serv2* it offers, this affects contract *Con2* as well as the IT system playing the role as its consumer.
- *Case 2:* Assume that the implementation of *TSV3* and *Serv3*, respectively, does not use any foreign services, or at least not the service installation *SI1* (to be shut down). Then, technical service version *TSV3* may remain unchanged and no modifications of contracts *Con3a* and *Con3b* are needed. Since the consuming IT systems are not affected by the described service shutdown, the change needs not be propagated on this path.

Usually, the information stored in contemporary SOA repositories is not detailed enough to be able to distinguish Cases 1 and 2. Hence, corresponding analyses might reveal more problems than actually exist. Opposed to this, the presented meta-model (cf. Fig. 1) allows distinguishing the two cases: Using relationship *usesTechnicalServiceVersion*, for example, it becomes possible to (exactly) detect the steps (*TechnicalActivity*) of a business process that invoke the technical service version to be changed or removed. With relation *belongsToTechnicalProcessVersion* the corresponding *TechnicalProcessVersion* (i.e., the executable business process) can be identified. If the latter is offered as a service, relationship *offeredAsTechnicalServiceVersion* refers to the corresponding *TechnicalServiceVersion*. Finally, if an IT system has a contract referring to the latter, it will actually be affected by the change; i.e. Case 1 applies and the analysis delivers the correct result. Compared to a SOA repository that solely contains information about the services consumed and offered by an IT system, higher quality in planning as well as decision support becomes possible.

Note that the information required by the SOA repository can be automatically gathered during the modeling of the business processes and services as well as the specification and implementation of the technical workflows and services (Buchwald et. al., 2012; Buchwald, 2012). Hence, only little additional effort becomes necessary for capturing repository data. For function-oriented SOA applications requiring no explicit process support, in particular, the described approach may be transferred to the modeling of services (e.g., based on UML), service implementations, and service consumers.

4.2 Impact of Future Changes

In general, the impact of a future change may depend on the exact time it will be applied. Hence, the time perspective needs to be considered in *what-if-analyses*. In particular, when simulating the impact of a change for multiple future points in time, different scenarios can be analyzed. In the following, we illustrate such time-aware *what-if-analyses*.

Assume that the sole service installation *SI0* of the technical service version *TSV0* shall be shut down (e.g., to reduce maintenance and operating costs). Assume further that the operator of *SI0* plans for a shutdown date in about 6 months from now. Hence, it shall be analyzed whether this date is advantageous from the perspective of the operator. As a first step, a graphical overview of all related contracts as well as their validity periods is created (cf. Fig. 4). Such a chart can be generated automatically based on the data stored in the SOA

[3] Without loss of generality, we assume that for a technical service version there exists only one service installation. Using the SOA repository data (i.e., relationship *hasInstallation*), it can be easily analyzed whether a service installation is actually available or a switch to an alternative service installation is possible; i.e. the ripple-effect ends at this point.

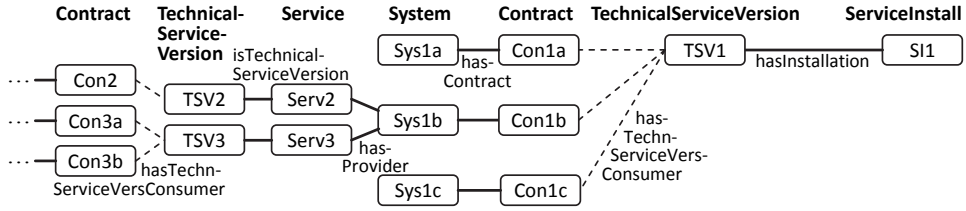

Figure 3: Sample data of the SOA Repository with Ripple-effect.

repository. The chart indicates that there are currently 4 contracts for *TSV0* and 4 IT systems depend on *SI0*. In 6, 10, 17, and 24 months (from now on), respectively, these contracts will reach their end date. Accordingly, these are candidate dates for a potential shutdown of *TSV0*; i.e., when reaching any of them, less IT systems will beaffected by the shutdown than before.

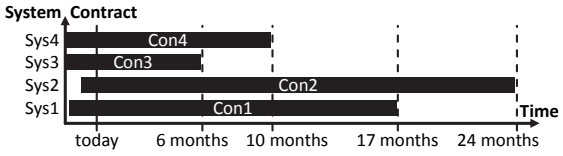

Figure 4: Graphical overview of the contracts for TSV0 and SI0 over time.

However, such an analysis does not take indirect dependencies (ripple-effect) into account, and hence may not return the complete set of IT systems actually affected by the service shutdown.

The impact graph depicted in Fig. 5a, in turn, shows these indirect dependencies. In particular, it visualizes each indirection level as a separate circle. Furthermore, all contracts valid at a future point in time are taken into account as well. Regarding the scenario from Fig. 5a, for example, all contracts that will still be valid in 6 months (from now on) are considered. Note that the depicted charts can be automatically generated based on the available SOA repository data.

When creating the chart from Fig. 5a, the dependencies depicted in Fig. 5b have been retrieved from the SOA repository: In addition to systems *Sys1*, *Sys2* and *Sys4*, whose dependency on *SI0* can be directly derived from contracts *Con1*, *Con2* and *Con4*, there exists an indirect dependency of *Sys5* on *SI0*. More precisely, IT system *Sys4* offers service installation *SI4* (cf. Fig. 5b), which is the sole implementation of the technical service version *TSV4*. Furthermore, the latter is consumed by *Sys5* (according to contract *Con5*). Consequently, the shutdown of *SI0* in 6 months might affect the four IT systems *Sys1*, *Sys2*, *Sys2* and *Sys4*. Note that this information can be used as a basis for discussing the intended shutdown with all persons responsible for the respective IT systems. Thus, the consequences of the service shutdown can be predicted and the required adaptations of the concerned IT systems be performed in advance. For example, the effort caused by these adaptations might be higher than the actual operating costs of *SI0*. Then, extending the operation period of *SI0* should be taken into account by decision makers.

The next candidate date for the shutdown of *SI0* will be in 10 months from today. Note that at this date, contract *Con4* will end and *Sys4* will no longer be affected by the service shutdown (cf. Fig. 4). The same applies to IT system *Sys5* since its dependency on *SI0* only indirectly exists via *Sys4*. This 10-month scenario is reflected by the impact graph depicted in Fig. 6, i.e., only two instead of four IT systems are affected by the shutdown of *SI0*.

Fig. 6b further shows that in 10 months there will be another service installation *SI0′* corresponding to the new technical service version *TSV0′*. In particular, the latter belongs to the same business service version *BSV0* as *TSV0*. Probably, there will be a high similarity between the two technical service versions. Hence, it might be an option to migrate the remaining contracts *Con1* and *Con2* to *TSV0′* and *SI0′* respectively. Due to the similarity of the two service installations *SI0* and *SI0′*, the technical realization of this change might require only little effort. Maybe it can even be realized without need to change the IT systems itself (e.g., by adapting the proxy (ESB) used for the service call (cf. (Buchwald et. al., 2011); (Buchwald, 2012)).

The date '10 months from today' might already be the optimal date for the shutdown of *SI0*. In order to verify this, further shutdown candidate dates (i.e., in 17 months and 24 months, respectively; cf. Fig. 4) should be analyzed in the same way.

In general, the presented *what-if-analyses* allow assisting SOA governance boards in making the right decisions, e.g., on whether a service shutdown shall be realized rather soon (resulting in lower operating costs) or, whether it makes sense from a business perspective to wait until a certain date

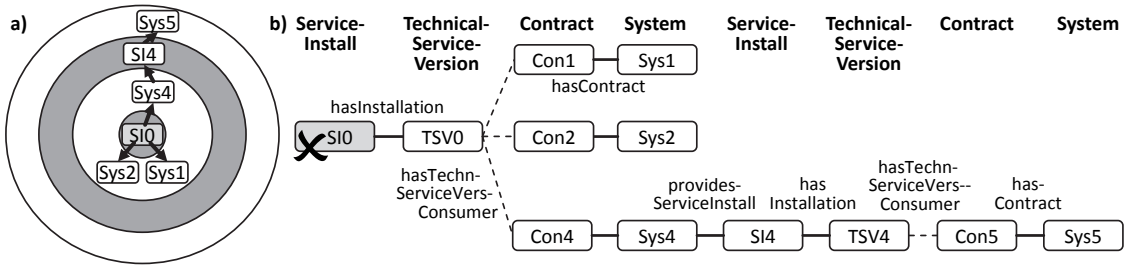

Figure 5: a) Impact graph and b) Dependencies from SI0 in 6 months.

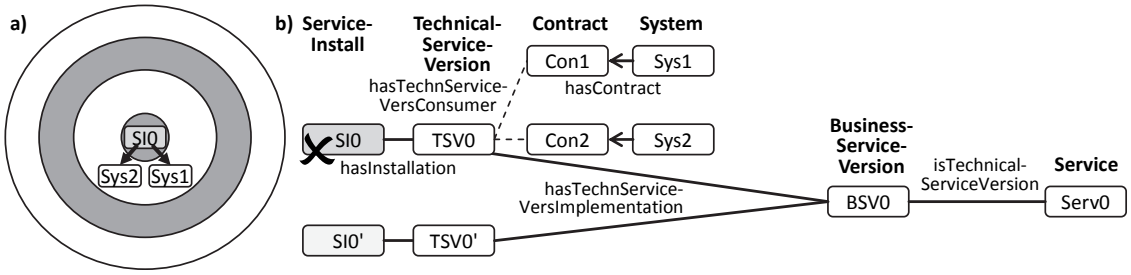

Figure 6: a) Impact graph and b) Dependencies from SI0 in 10 months.

(resulting in lower migration costs). For further scenarios utilizing what-if-analyses we refer interested readers to (Tiedeken, 2010).

5 DISCUSSION

Except our previous work (Buchwald et. al., 2010; Buchwald, 2012; Tiedeken, 2010), we are not aware of more advanced approaches dealing with the design of a SOA repository as described in this paper. The same applies to the presented *as-is-analyses* and *what-if-analyses* based on SOA repository data. When taking a broader view on the general topic, however, few related approaches can be identified.

(Xiao et. al., 2007) address changes of business processes in the context of a SOA. In addition, the propagation of changes to dependent artifacts (i.e., ripple-effect) is considered. When a change of a business process must be performed, for example, it is first determined which concomitant changes of dependent business processes become necessary. In turn, this analysis is based on a classification of process change types (e.g., inserting an activity) as well as their effects. Furthermore, cascading service invocations are analyzed in order to be able to identify the services indirectly affected by the original change as well. The approach suggested by (Xiao et. al., 2007), however, does not include the calculation of such dependent objects (e.g., based on SOA repository data as in our approach). Instead, it focuses on the estimation of the costs created by the necessary adaptations of program source code.

(Wang et. al., 2010) presents a classification of both process and service changes. The ripple-effect is considered as well. When a specific process change shall be applied, for dependent services, it is determined which kind of service changes are concomitantly required. In turn, the latter might necessitate changes of the business processes consuming these services. Altogether, the focus of this approach is on evolving business processes in a SOA and the analysis of respective process and service structures. Data offered by a SOA repository is not taken into account for these analyses.

There exist several standards as well as commercial products for realizing SOA repositories. In the following, we presented selected ones. In addition to the functions they offer, we discuss to what extend they may serve as basis for realizing the presented meta-model as well as the described analyses.

UDDI (OASIS, 2002) is a directory service standardized by OASIS, which especially allows for look-ups of service endpoints. However, its underlying meta-model is very limited and does not allow for any extensions (i.e., new object or relationship types cannot be defined). However, user-defined categories can be created (tModel) and may then be used to assign services to categories. Except few predefined queries, no analyses are provided.

ebXML defines standards for exchanging business data. It includes the ebRIM (Registry Information Model) as defined in (OASIS, 2005). The corresponding meta-model, however, only allows for the storage of the technical perspective. The business perspective of service objects and contracts, in turn, is not covered by this standard. The same limitations exist in respect to analyses capabilities.

USDL (Oberle et. al., 2012) is an emerging standard published by W3C. Its goal is to comprehensively describe services including the business perspective; i.e., in addition to technical meta-data of the services, operative as well as economic data are covered (e.g. availability and pricing). *As-is-* and *what-if-analyses* are not considered.

IBM offers the commercial SOA repository WSRR, which includes a "Basic Governance Profile" (Sachdeva, 2007). In particular, the latter allows storing both technical and business objects. The business perspective, however, is very limited compared to our SOA repository approach. More precisely, a (business) description may be defined for services, but not a business specification of their operations (including parameters). However, WSRR is extendible regarding its object and relationship types. Consequently, WSRR could be used as a basis for realizing our meta-model. Regarding the presented analyses, however, WSRR offers only limited support (e.g., visualizing dependencies). However, user-defined analyses may be added as extensions.

CentraSite Active SOA (Rogers, 2006), a product offered by Software AG, looks similar in respect to the object types, extensions and analyses it offers. However, the latter only allow for interactively visualizing repository objects and their dependencies.

Oracle Enterprise Repository (Oracle, 2008) as well as HP SOA Systinet (Hewlett-Packard, 2010) are based on UDDI. Both allow storing additional object types compared to the ones defined by the UDDI standard. This allows managing business objects as well as contracts. In addition, the visualization of dependency charts as well as some limited analyses are supported.

The ARIS repository (IDS, 2008) enables the storage of SOA artifacts as well. Since it maintains the database of a business process modeling tool, there exist significant differences compared to the products mentioned above. In particular, ARIS repository focuses on the business instead of the technical perspective. For example, it is not possible to use the ARIS repository during a service call (i.e., at run-time) for a service endpoint look-up. As an advantage, large numbers of object and relationship types (cf. ARIS models), which belong to the business perspective, may be stored. Furthermore, ARIS supports repository analyses with restricted functionality. Similar to other products, they may be extended by additionally implemented analyses (see Buchwald (2012) for some examples of extensions).

Altogether, no standard or product is equipped with a meta-model that completely covers the object and relationship types presented in this paper. Instead existing approaches focus either on the technical or the business perspective solely. Regarding *as-is-analyses*, existing products only provide a very limited functionality, whereas *what-if-analyses* are not covered by them at all. Since the meta-model and the analyses algorithms of existing commercial products may be extended, however, it is possible to use these products as basis for implementing of concepts and techniques presented in this paper.

6 SUMMARY AND OUTLOOK

We presented a comprehensive meta-model for a SOA repository enabling various analyses. *As-is-analyses* check whether any problem occurs when considering the current repository data (e.g., contract of a service with missing service installation). In turn, *what-if-analyses* allow simulating the consequences of future changes. In this context, the propagation of problems (ripple-effect) is considered as well. Overall, our approach significantly enhances existing SOA repository standards and products.

In principle, the extensibility features provided by certain SOA tools, allow realizing the presented concepts. Ourselves, we have validated our meta-model as well as the related analyses by implementing a powerful proof-of-concept prototype based on a relational database system (Tiedeken, 2010).

So far, we have neither implemented the entire meta-model nor the analysis algorithms based on any commercial SOA repository. If such an implementation was available, our approach would be validated in a real-world case study. Finally, SOA repository data as well as related analyses should be integrated into SOA governance processes.

REFERENCES

Abrial, J.R., 1974. Data Semantics. In *Proc. IFIP Working Conf. Data Base Management*, pp. 1-60.

Bauer, T., Buchwald, S., Reichert, M., 2013. Improving the Quality and Cost-Effectiveness of Process-Oriented, Service-Driven Applications: Techniques for Enriching Business Process Models. In *Ramanathan, R., Raja, K., (eds). Service-Driven Approaches to Architecture and Enterprise Integration. Information Science Reference*, Hershey, 104-134.

Buchwald, S., 2012. *Increasing Consistency and Flexibility of Process-oriented Applications by Service-orientation*. PhD thesis. University of Ulm (in German).

Buchwald, S., Bauer, T., Reichert, M., 2011. Flexible Process Applications in Service-oriented Architectures - An Overview. *EMISA Forum*, 31(3) (in German).

Buchwald, S., Bauer, T., Reichert, M., 2012. Bridging the Gap Between Business Process Models and Service Composition Specifications. In *Lee, J., Ma, S.-P., Liu, A., (eds). Service Life Cycle Tools and Technologies: Methods, Trends, and Advances*. IGI Global, Hershey.

Buchwald, S., Tiedeken, J., Bauer, T., Reichert, M., 2010. Requirements of a Meta-model for SOA Repositories. In *Proc. 2nd Central-European Workshop on Services and their Composition*, 17-24 (in German).

Erl, T., 2005. *Service-Oriented Architecture – Concepts, Technology, and Design*. Prentice Hall.

Hewlett-Packard Development Company, 2010. *HP SOA Systinet Software Data Sheet*.

IDS Scheer, 2008. *ARIS SOA Architect: Business Processes as the Basis for Service-Oriented Architectures*.

Josuttis, N., 2007. *SOA in Practice - The Art of Distributed System Design*. O'Reilly.

OASIS, 2002. *Universal Description, Discovery, and Integration (UDDI). Version 3.0*.

OASIS, 2005. *ebXML Registry Information Model. Version 3.0*.

Oberle, D., Barros, A., Kylau, U., Heinzl, S., 2013. A unified description language for human to automated services. *Information Systems*, 38 (1), 155–181.

Oracle, 2008. *Oracle Enterprise Repository and Oracle Service Registry for the SOA-Lifecycle*. Oracle Data Sheet.

Rogers, S., 2006. *CentraSite: An Integrated SOA Registry and Repository*. White Paper, Software AG.

Sachdeva, N., 2007. *Customize the WebSphere Service Registry and Repository Governance Profile*. IBM developerWorks.

Tiedeken, J., 2010: *Concept and Realization of a Logically Central SOA Repository*. Master's thesis. University of Ulm (in German).

Wang, Y., Yang, J., Zhao, W., 2010. Change Impact Analysis for Service based Business Processes. In *Proc. Int. Conf. on Service-Oriented Computing and Applications*, IEEE.

Xiao, H., Quo, J., Zou, Y, 2007. Supporting Change Impact Analysis for Service Oriented Business Applications. In *Proc. Int. ICSE Workshop on Systems Development in SOA Environments*, IEEE.